SCAR: A Stateless Approach to Achieving Scalable Quality of Service

D. Reid and M. Katchabaw
Department of Computer Science
The University of Western Ontario
London, Ontario, Canada N6A 5B7
{dreid28, katchab}@csd.uwo.ca

***Abstract*—*Previous attempts at providing widespread Quality of Service (QoS) for the Internet have met with only limited success. Integrated Services and Differentiated Services, the two most popular architectures proposed, suffer from scalability and flexibility concerns respectively. More recently, numerous other architectural proposals have been introduced, but have also been met with limited success. This paper introduces a prospective new approach which addresses several issues others have failed to solve effectively. This approach, SCalable Aggregate Reservations (SCAR) is highly scalable, offers additional functionality, and is quite flexible and robust in supporting QoS for networked applications.*

***Keywords**: QoS, aggregate reservations, stateless networking*

1. Introduction

Originally designed as a best-effort service, the Internet has grown significantly, and so have its needs. Today, many applications require a service level better than best-effort can offer. In effect, these applications require a high level of Quality of Service (QoS), which is an expression used to depict the overall experience a user or application receives over a network [8]. Over the Internet, this QoS is typically measured in terms of bandwidth, loss, delay, jitter, and availability.

QoS models strive to take a best-effort network and transform it into one which can provide bandwidth and delay assurances to its applications. There have been several fields of thought on providing QoS. By far, the two most popular and accepted philosophies are the Integrated Services Model (IntServ) [7] and Differentiated Services Model (DiffServ) [2]. However, neither IntServ nor DiffServ have gained widespread acceptance due to scalability and flexibility concerns respectively. Several lightweight protocols have emerged since, including SRP [1], FIRST [5], SCORE [12], SGS/SOS [9], YESSIR [10], and many others. These approaches, however, generally lack the functionality and acceptance needed for an integrated services network, or have drawbacks making them difficult to deploy or use.

This paper presents a stateless QoS architecture for the Internet that can provide end-to-end QoS guarantees to multiple flows on demand. The architecture, called SCalable Aggregate Reservations (SCAR), achieves scalability by aggregating flows into predefined classes. It requires little processing at intermediate nodes and can additionally enable billing functions, security functions, and mechanisms to achieve optimal resource allocation [11]. SCAR belongs to a variant of the IntServ philosophy; one that aims to provide the same flexible service, but in a stateless network environment. This stateless property is one which addresses scalability concerns that have arisen with traditional IntServ. SCAR combines the scalability of DiffServ and the functionality of IntServ into one flexible package.

A new signaling protocol, the SCAR Signaling Protocol (SCAR-SP), was developed to enable signaling within this new architecture. Its reversible design allows either the sender or receiver to make reservations. It presents many additional advantages, including the ability for senders to specify their unique traffic characteristics, and a soft-state approach which can remove reservations if not properly terminated. Experimentation has shown that nodes can handle hundreds of thousands of flows simultaneously with little impact on local node performance.

The remainder of this paper is organized as follows. Section 2 presents the architectural design of the SCAR model, the signaling protocol SCAR-SP, and scheduling in SCAR. Section 3 discusses experimental results and comparative analyses with other approaches. Section 4 concludes this paper with a summary and discussion of future work in this area.

2. The SCAR Approach

In this section, we present our approach, SCalable Aggregate Reservations (SCAR). We discuss its architectural design and signaling protocol SCAR-SP, and conclude this section with a discussion of scheduling and policing.

2.1. Architectural Design

SCAR was designed to meet the following goals:

Scalability – Traditionally, the amount of router state increased proportionately to the number of individual flows with the IntServ model, leading to serious scalability concerns. SCAR aggregates individual flows into classes, requiring no per-flow state in the intermediate nodes. Additionally, techniques to insert state into the packets themselves, as in SCORE, are not required.

Simplicity – To maintain simplicity and reduce overhead, SCAR requires only minimal state in nodes to keep track of aggregate classes only, and does not support multicasting, in line with recent thoughts on the subject [4]. The need for resource "garbage collection" as in SGS or node synchronization, as some other solutions require, is also unnecessary.

Robustness – Mechanisms are in place to account for reservation failure, routing changes, and message loss. The onus has been placed on the end-host to ensure messages and reservations are timed properly to account for these failures. Intermediate nodes are able to verify traffic specifications, and police those who are not adhering to their contract.

Flexibility – Less state typically means less functionality and flexibility. SCAR can provide bounds on both delay and bandwidth and enable optimal resource allocation without the need for per-flow state.

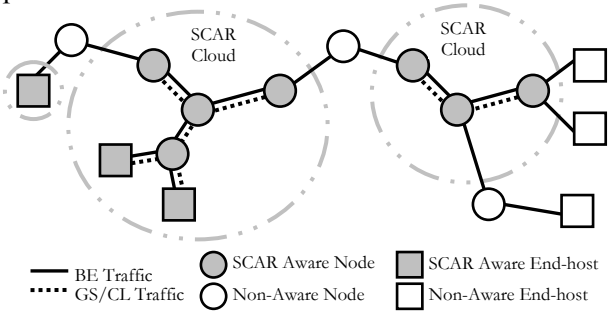

Figure 1. Logical separation of a network

For the time being, we consider in this paper only three classes: guaranteed service, controlled-delay, and best-effort service. Other predefined classes such as controlled-load or custom classes are quite possible to incorporate into SCAR, yet we consider for all intents and purposes a three class architecture including the guaranteed service class which can provide the most stringent of QoS guarantees, and the controlled-delay class for delay-insensitive applications. To provide these classes within an existing best-effort network, we conceptualize a logical partition. By separating link speed between each class, and with proper management, a guaranteed service network can be placed atop the existing best-effort network. This abstraction can be seen in Figure 1. To distinguish between classes, data packets are classified within the headers through marking.

In Figure 1, a logical separation can be seen between all SCAR aware nodes and end-hosts. Operation of SCAR can still take place transparently within non-aware areas of the network, although QoS cannot be guaranteed in this case when heavy congestion occurs at non-aware nodes. (End hosts will, however, will be made aware of any transparent operation through non-aware nodes, and can cancel their reservation at any point.)

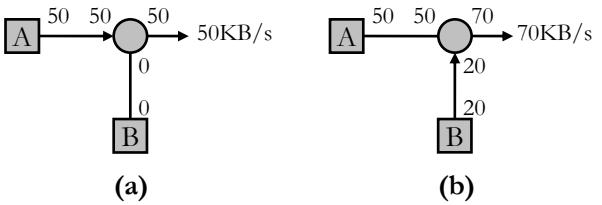

Figure 2. Additive property of aggregates

Using the additive properties of aggregation, bandwidth guarantees can be made without the need for per-flow maintenance. This is demonstrated in Figure 2, where end-host A creates a 50KB/s reservation in (a), followed by a 20KB/s reservation by host B in (b). Aggregate state is maintained on all incoming and outgoing links. This simplicity, coupled with proper policing, enables powerful network services.

There are several mechanisms which need to be in place in a SCAR network, as shown in Figure 3. Traffic must be classified, shaped, and policed before the egress link with non-conforming packets being isolated to ensure fairness to other flows. This can potentially be done through either re-marking or dropping. Proper scheduling must additionally be in place to provide accurate

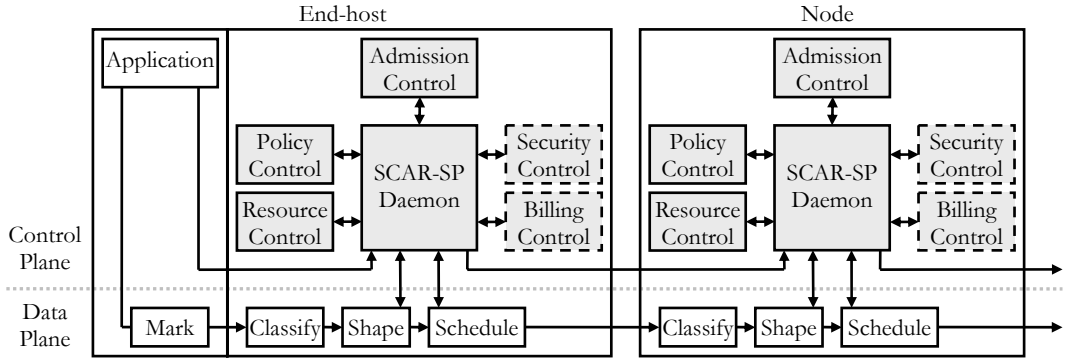

Figure 3. SCAR architecture

bandwidth and delay bounds. SCAR does not specifically dictate the scheduling techniques used to provide bandwidth and delay guarantees in nodes, as there are already approaches to do so. Further details on the SCAR architecture can be found in [11].

2.2. The SCAR-SP Signaling Protocol

SCAR-SP has been designed to provide efficient call mechanisms when creating, refreshing, or removing reservations. Unlike most signaling protocols, which are restricted to either a sender or receiver-oriented design, SCAR-SP enables both approaches. Either the receiver or sender can be given the responsibility to provide the flow specifications for the reservation. In many cases, receiver control is favoured over sender controlled, as senders are often not aware of what resources are available to the receiver, including bandwidth or even computational power. On the other hand, receiver control can also be a hindrance, especially in the case where senders are billed for QoS, or the sender wants more control over policy.

Another advantage of SCAR-SP's design is that senders are first responsible for characterizing the traffic it will be transmitting. By knowing the traffic model, receivers establishing a reservation can make more informed decisions. For example, if a receiver is not aware that the sender will be transmitting at a variable rate, it may unknowingly make a constant rate reservation, in effect causing underutilization. Additionally, if the sender is able to characterize its traffic in more than one way, the receiver can then be given the option to choose which specification it wishes to receive.

SCAR-SP messages require state typically not associated with signaling messages to be inserted within them. This state is minimal, and cannot be avoided in soft-state architectures. It should be reiterated that this type of state is significantly different from the state associated with IntServ scalability concerns. The largest advantage to keeping this state in the signaling protocol is the fact that no lookup is required on behalf of intermediate nodes.

SCAR-SP is a soft-state protocol which provides better dynamic adaptability and robustness than a hard-state protocol, and allows adaptation of routing changes to take place fairly seamlessly with end-host cooperation. Unlike RSVP where refresh messages are generated by intermediate nodes hop-by-hop, SCAR-SP shifts this responsibility to the end-hosts in an end-to-end fashion. They are then required to initiate periodical refreshes so that their soft-state reservations are kept active. This further reduces processing overhead at intermediate nodes.

If a reservation expires, resources are automatically de-allocated. To maintain simplicity, SCAR-SP is simplex and can only offer unidirectional reservations. To establish a bidirectional flow, two separate reservations must be made. SCAR-SP messages are sent out-of-band over the control plane, which is logically separated from the data plane. Its design can also handle lost messages, and requires minimal processing power at the nodes.

The operation of SCAR-SP is similar to most other signaling protocols. The SCAR-SP daemon, shown earlier in Figure 3 facilitates the creation and removal of reservations in conjunction with policy and admission control modules. Admission control determines if enough resources are available to admit a reservation, while policy control determines if the end-host is permitted to do so. The classifier determines which class the traffic belongs to; either the guaranteed service class, or best-effort. Policing and shaping of the guaranteed service class ensures traffic is behaving properly, sending at proper peak and average rates.

The scheduler can then guarantee the reserved bandwidth and delay to this traffic.

In an aggregated architecture, admission control is rather simple; only the current and maximum allowed aggregate reservation need be known. The maintenance of this aggregate, however, is far more difficult as signaling loss and partial reservations are introduced. Part of the difficulty arises due to the fact that admission control is based on an end-to-end transaction. That is, if all intermediate nodes along the end-to-end path do not accept the reservation, then all nodes which have since accepted it locally must roll-back to a previous state. Likewise, lost signaling messages can create confusion as partial reservations occur. Normally this would imply that nodes must maintain internal state. However, SCAR-SP can deal with these problems effectively [11].

SCAR-SP defines four phases of operation; discovery, reserving, refreshing, and tearing down. The discovery phase entails determining the current status of the end-to-end reservation path. The reservation phase involves the merging of flows in to a service class aggregate, while the refresh phase keeps these reservations active. In the teardown phase, a reservation is explicitly de-allocated. Three messages are defined; discover (DISC), reserve (RESV), and refresh (REFR). DISC messages are used in the discovery phase, while RESV messages are used during the reserving phase. REFR messages are used during the refreshing and teardown phase and use expiration identifiers to track allocations and deallocations appropriately. Further details on SCAR-SP can be found in [11].

2.3. Scheduling and Policing

By controlling link bandwidth and buffering at each node, packet loss, delay, and throughput can be managed. This can be accomplished through admissions procedures allowing access to the resources, and scheduling disciplines to limit the competition of flows and to balance allocated bandwidth against delay requirements to meet the needs of reserved flows [11].

To provide node-local QoS guarantees to a flow, a packet scheduling discipline is used to guarantee bandwidth and place an upper bound on delay. The discipline may additionally provide bounds on jitter or loss. This is done by choosing which packet to transmit when its respective outgoing link becomes idle.

While mechanisms within a node must be in place to address local QoS guarantees, there must also be mechanisms to ensure the end-to-end QoS is met. Due to the uncertainness of packet switching networks in a multiplexing environment, traffic patterns can become distorted through differing areas of load on the network. These distortions can result in bursts of traffic at differing points, regardless of how the traffic had entered the network, and can compromise end-to-end QoS metrics including packet jitter, delay, and possibly loss. Consequently, appropriate mechanisms must be employed to handle these situations.

The SCAR approach, as discussed earlier, is flexible and can support a wide variety of scheduling disciplines and algorithms that provide guarantees on bandwidth and delay. Since extensive work has already been dedicated to the development of approaches to scheduling network resources, this allows implementations of SCAR to leverage this existing work, and make use of an approach appropriate to the situation.

Either unintentionally or maliciously, the possibility exists that flows will not adhere to SCAR conventions. A misbehaving flow within an aggregate can not be directly policed, as no state is maintained to identify it. Instead, misbehaving aggregates are policed by isolating them from the network. Thus, the further the architectural edges are pushed towards end hosts, the greater the isolation granularity.

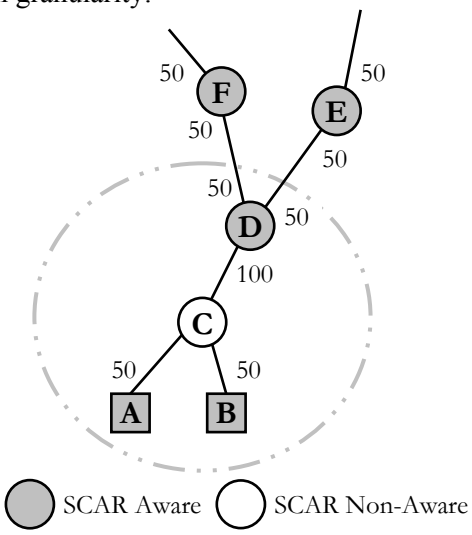

Figure 4. Policing in SCAR

In Figure 4, a network is shown in which the gateway, depicted as node D, and end-hosts A and B are SCAR aware. Node C located between the gateway and end-hosts is not SCAR aware. In this

particular case, hosts are still able to make reservations transparently through node C, and, as long as there is no congestion at this node, QoS should not be compromised. In this case, both A and B have 50KB/s reservations to different destinations. A problem exists, however, if host A or B decides to transmit at a higher rate than reserved, or maliciously floods the network with packets marked as reserved. As there is no state maintenance at node D, it is therefore not aware who the perpetrator is, except that it is coming from the direction of node C. It must then police this aggregate, resulting in service degradation to both hosts A and B. If node C was SCAR aware, it could have policed only the misbehaving portion of the aggregate and not both A and B necessarily.

Policing entails that each node will simply prevent any packets exceeding the reserved aggregate from obtaining QoS, with the onus placed on end-hosts to never exceed their reservation. Nodes may allow some tolerance to over-utilization, remark packets into a best effort class, or drop packets to ensure conformance.

The issues of scheduling and policing in SCAR are discussed further in [11], with the various scheduling options available described in detail.

3. Experimental Results and Analysis

In this section, we will analyze SCAR and its signaling protocol, SCAR-SP. Where possible, we also compare our approach to current RSVP implementations, and other signaling protocols and architectures mentioned in Section 1, including Boomerang, SGS, and YESSIR. With no implementation available for other systems such as SCORE, we were unable to assess and compare with their performance. For brevity, we report on highlights of the most important and interesting experimental results; for complete results, the reader is urged to consult [11] for details.

3.1. Experiment Design and Setup

Unfortunately, there is a lack of consistency in proof of concept and implementation of work in this area. RSVP has multiple competing implementations available, while other approaches have implementations for only specific configurations of Linux or BSD Unix. Some of the newer approaches have been implemented for simulation packages like the Network Simulator ns-2 [6]. As a result, analyses of SCAR involved a mixture of live network tests and simulations, with baseline testing to ensure that results were consistent and comparable, as discussed in [11].

For primary testing, a 400MHz Celeron workstation with 256MB of RAM was used, as this most closely matched the configurations of test machines in related work (again to ease comparisons). As an operating system, we used the Knoppix Linux Live CD 3.6 distribution with kernel 2.4.27. By doing so, there was no need for swap space or additional overhead as the lightweight kernel is loaded directly into RAM. When traffic generation was required, a collection of Athlon XP 1800 machines with 512MB of RAM were used, with the same Linux operating system, connected by 100MB Ethernet.

For simulations, we used version 2.27 of ns-2. While there are some limitations when measuring nodal performance metrics, ns-2 is quite useful in generating very large topological simulations for study. Nodal performance was measured through software instrumentation, as ns-2 is event driven, and runs independent from processing power. We used the Celeron for our simulation system as well.

SCAR and SCAR-SP were implemented and tested in this simulation environment. To implement these in ns-2, sender and receiver agents were first created to generate and handle reservation requests and responses. The sender agent was additionally responsible for generating and marking flows, and initiating periodic refreshes by sending DISC messages.

3.2. Validation of SCAR and SCAR-SP

Before conducting performance and scalability testing, we first carried out a series of simulations to validate SCAR and SCAR-SP and ensure that they were functioning correctly. This involved the creation, maintenance, and teardown of flows, the proper scheduling of flows, and tests of flow protection, admission control, and the policing of misbehaving flows. In all cases, SCAR and SCAR-SP performed as they should and were deemed to be delivering their respective functionality correctly.

3.3. Performance and Scalability Tests

Many signaling protocols and architectural designs have been slow to gain footing due to scalability concerns, which is defined as the capacity for the network to expand the amount of flows, nodes, and traffic. These concerns are a

direct result of the overhead required when signaling tens, if not hundreds, of thousands of flows simultaneously within core nodes.

3.3.1. Processing Overhead

Processing overhead was calculated as the percentage of time spent by the test system in processing messages and maintaining and scheduling flows. This data was collected using tools such as *top* and *tcpdump* when live tests were possible, and through instrumentation when simulations were required.

For test purposes, the reservation refresh interval was set to 30 seconds in all cases as suggested in [3], and all flows are set to live for 62.5 seconds before being regenerated, a value selected for comparison purposes with previous work in this area. While flows in the real world will most likely live for longer periods, we choose a shorter interval to address scalability concerns during periods with many short-lived reservations. In most cases, a session within this interval would include a setup, two refreshes, and a teardown.

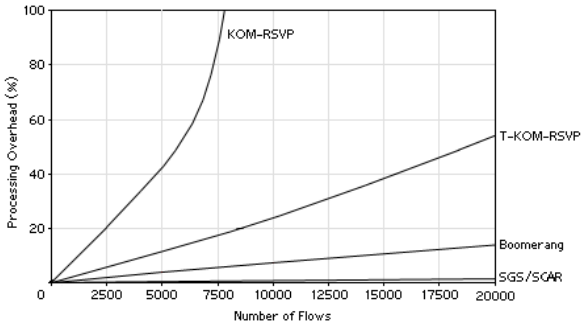

Figure 5. Processing Overhead

We summarize the overhead results in Figure 5. As can be seen, implementations of RSVP (KOM-RSVP, and the more optimized T-KOM-RSVP) had the most overhead, while SCAR and SGS produced excellent results, with SCAR just edging out SGS. Both SCAR and SGS could handle 100,000 simultaneous flows with less than a 50% processing load (with SCAR at 47.0% and SGS at 48.7%). Boomerang performed reasonably well, but we could not observe YESSIR in this case due to issues in generating compatible traffic.

3.3.2. Memory Overhead

Memory overhead at intermediate nodes is largely a function of the number of flows reserved and maintained at the node. Architectures and protocols that require more memory run the risk of exhausting resources on nodes, particularly as the number of flows increases. Memory requirement measurements were obtained from monitoring the */proc* file system in testing live implementations, through instrumentation of simulations, and also through the examination and analysis of source code if experimentation proved difficult.

Memory overhead requirements for various approaches are shown in Figure 6. RSVP implementations had the highest memory overhead; the ISI RSVP implementation was also examined, but found to be quite limited in the number of flows that could be supported. SCAR and SGS both had low memory needs, while Boomerang and YESSIR had moderate needs. SCAR is quite scalable, as the memory it requires is independent of the number of flows due its stateless aggregate philosophy.

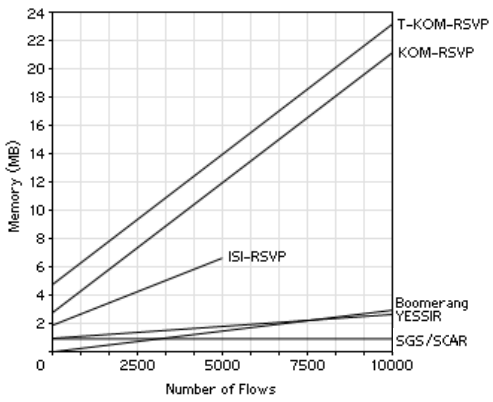

Figure 6. Memory Overhead

3.3.3. Network Overhead

Network overhead refers to the amount of additional network traffic introduced to support the creation, maintenance, and teardown of flows. Since SCAR requires state inserted in signaling messages, it incurs a higher overhead than other approaches as shown in Figure 7.

Network overhead was computed based on an analysis of the protocol messages required to manage the number of flows required, assuming 30 seconds between refreshes and 20 hops end-to-end. It is important to note that since YESSIR is an in-band protocol, it adds little network overhead itself; the overhead reported is primarily that of RTCP used to carry it.

The higher network overhead for SCAR does not cause a significant concern. As shown earlier, this network overhead does not impose a significant

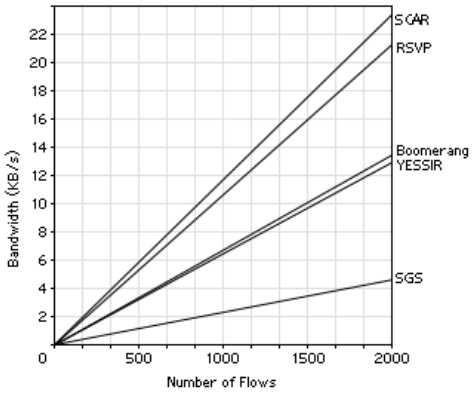

Figure 7. Network Overhead

processing or memory burden on intermediate nodes. Furthermore, the signaling overhead for SCAR is minimal in comparison to the network requirements of the flows themselves. For example, typical rates on Sprint backbones are 2488Mbps (OC-48) with the number of flows for any given minute in the area of several hundred thousand [7]. Signaling overhead for SCAR-SP on such a link serving a generous 100,000 reserved flows requires only 0.3% of its link capacity. In fact, network overhead remains a linear function, and it is generally not thought of as a problem when scaling to large numbers of flows in a call admission organization. When one considers that processing and memory considerations would prevent many approaches, including RSVP, from being able to handle this number of flows in the first place, SCAR is performing reasonably well.

4. Conclusions and Future Work

This paper presented SCAR, a stateless QoS architecture for the Internet that can provide end-to-end guarantees to multiple flows on demand. It is highly scalable in its memory requirements, network requirements, and most importantly, processing requirements. It provides a greater set of functionality than previous lightweight protocols and also provides extended billing, pricing, and security modules, and mechanisms to achieve optimal resource allocation. It is not dependant on any particular scheduling algorithm, and can handle multiple traffic classifications.

A new soft-state signaling protocol, SCAR-SP, was presented to enable signaling within this new architecture. Its design allows either the sender or receiver to make reservations, and allows senders to specify their unique traffic characteristics to the network. It can additionally operate through non-aware intermediate nodes in a transparent operation. Experimentation has shown that nodes can handle hundreds of thousands of flows simultaneously with little performance impact.

In the future, we plan to continue experimentation and port our ns-2 implementation for use in live testing. We are also currently investigating a variety of competing techniques to reduce the network overhead in signaling in SCAR-SP. As mentioned earlier, more work needs to be done in developing billing and security modules for SCAR, continuing the work in [11]. Additionally, optimization of resource allocations, which has rarely been addressed in the past, needs to be researched and tested more thoroughly.

References

[1] W. Almesberger, et al. SRP: A Scalable Resource Reservation Protocol for the Internet. *Computer Communications, Vol 21, Number 14*. Nov. 1998.

[2] S. Blake, et al. An Architecture for Differentiated Services. *RFC 2475*, Dec. 1998.

[3] R. Braden, et al. Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification. *RFC 2205*, Sept. 1997.

[4] I. Brown, et al. Internet Multicast Tomorrow. *The Internet Protocol Journal, Volume 5*, December 2002.

[5] T. W. K. Chung, et al. Flow Initiation and ReServation Tree (FIRST): A New Internet Resource Reservation Protocol. In Proc. IEEE 1999 Pacific Rim Conf. on Communications, Computers and Signal Processing, Victoria, Canada, 1999.

[6] K. Fall, K. Varadhan. The ns Manual. *Reference documentation*. Mar. 2005.

[7] C. Fraleigh, et al. Packet-Level Traffic Measurements from a Tier-1 IP Backbone. *Sprint ATL Technical Report TR01-ATL-110101*. Nov. 2001.

[8] Nortel Networks. Introduction to Quality of Service. *Nortel Networks White Paper 56058.25_022403*. Mar. 2003.

[9] E. Ossipov, G. Karlsson. SOS: Sender Oriented Signaling for a Simplified Guaranteed Service. *In Proc. of Third International Workshop on Quality of Future Internet Services*, 2002.

[10] P. Pan, H. Schulzrinne. YESSIR: A Simple Reservation Mechanism for the Internet," *Proc. International NOSSDAV Workshop*. Cambridge, United Kingdom, July 1998.

[11] D. Reid. SCAR: A Stateless Architecture for Achieving Scalable QoS. *Masters Thesis. Department of Computer Science, The University of Western Ontario*. May 2005.

[12] I. Stoica. Stateless Core: A Scalable Approach for Quality of Service in the Internet. *PhD Thesis, CMU*, 2000.